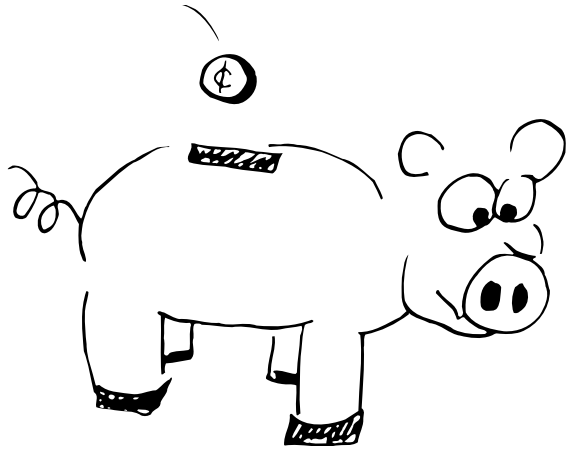
28 Accounting



28.1 INTRODUCTION

Accounting is an anachronism. Back in the good old days of expensive timesharing hardware[1], the use of accounting was common. Today, it is used primarily to provide an audit trail for tracking security break-ins, and perhaps to monitor disk and printer usage or to convince management of the need for more resources.

The UNIX kernel and various system programs keep accounting records for CPU time, login sessions, printer usage, modem usage, and a number of other system resources. A system administrator must wade through the megabytes of collected data and decide what to archive, what to keep temporarily, and what to throw away.

Accounting data files can grow quickly and will overflow the disk if not truncated regularly. Management of log files in general, including accounting files, is covered in Chapter 12, *Syslog and Log Files*.

The disk space consumed by these data files is one of the main costs of accounting. Another potential cost is reduced system performance. In particular, process accounting requires an additional write to disk for each process. On some older systems this seemed to have a perceptible effect, but on modern systems the impact is negligible.

1. A one-MIPS Vax 11/780 cost between $200,000 and $300,000 in 1980.

The accounting systems are quite different under ATT and BSD, but both systems measure essentially the same data. The BSD system has a few C programs that summarize single quantities of interest, while the ATT system is built on shell scripts that collect and summarize the data into comprehensive reports. System administrators tend to write scripts that take the output of the available tools and convert it to a format that suits their needs.

28.2 WHY BOTHER WITH ACCOUNTING?

The most obvious reason for accounting is to bill for resources used. This is appropriate for commercial providers, but not usually for in-house installations. Government contractors may be required to keep detailed accounting information.

Universities that purchase computers with federal research dollars are often required by their funding agencies to keep accounting records if they charge research grants for use of the computing infrastructure. In the past it was common to charge a "sales tax" based on usage. But as accounting fell out of favor and the volume of data saved to protect against a possible audit became unmanageable, many sites changed their charging algorithm to an "income tax" on research grants, thus sidestepping accounting altogether. This approach has even survived a couple of audits.

See Chapter 23 for more information about security.

The most compelling reason to maintain accounting records remains the discovery of unauthorized use. Hackers[2] are out there, and accounting records are often the only way a site can tell if it has been attacked.

28.3 WHAT TO MEASURE AND ARCHIVE

Connect-time records written by **`login`** to the **`wtmp`** file are often the most helpful accounting records for tracking misuse. CPU accounting information written by the kernel to the **`acct`** or **`pacct`** file is of less value, because only the names of commands are recorded, not their arguments. Printer accounting is useful if you either charge for printing or provide a "reasonable" amount of printing without charge. Records of disk usage are used to browbeat users into cleaning up their files.

Before truncating accounting files, you may want to create a permanent record by archiving summaries to disk or tape. Raw accounting data files, summaries of their contents, and some log files may all need to be archived if records are required for a possible audit.

2. Or more accurately, "crackers." A person that breaks into a computer system used to be called a hacker, but these days "hacker" just as often denotes a prodigious, but possibly inelegant, coder. Technical lexicographers had to come up with a new word for the bad guys. For example, "Stay away from my network, you honky cracker!"

Four broad philosophies exist for archiving these files:

Conservative	Archive all accounting files and most log files to tape. Store tapes in a secure location.
Sensible	Archive summary accounting files to tape or disk. Rotate other accounting files, overwriting them as the rotation sequence wraps around. Keep at least a month's worth of raw data to facilitate tracking if security problems arise.
Carefree	Restart all accounting and log files periodically and throw away the old ones.
None	Turn off accounting as much as possible and truncate files nightly out of **cron**.

Choose a philosophy that fits your site's requirements.

28.4 ACCOUNTING IN A NETWORKED ENVIRONMENT

In an environment with lots of workstations on a local area network, separate accounting on each workstation can be a real nightmare. Programs run out of **cron** can collect and summarize data locally; if further summaries by host are required, you must build a tool that can be run from a central host.

An example of such a tool (for printer accounting using BSD's **lpr** system) is a pair of scripts we call **pachelper** and **pacmaster**, which are included on the CD-ROM.

28.5 ACCOUNTING UNDER BSD

In BSD, accounting files were traditionally located under **/usr/adm**. A few years ago, Sun reorganized the filesystem and introduced the **/var** directory for files that vary on a per-host basis. This allowed directories such as **/usr** to be made machine-independent. With this organization, local files are concentrated in only a few places rather than being scattered randomly throughout the filesystem.

Many vendors have followed Sun's lead, and accounting files now often live in **/var/adm**. As an aid to old-timers and old software, there is often a symbolic link from **/usr/adm** to **/var/adm**.

CPU Accounting

CPU accounting must be enabled in the kernel, and then can be turned on by the **accton** command. **accton** *filename* turns accounting on (*filename* must identify an existing file) and **accton** with no arguments turns accounting off.

accton is usually executed at boot time from one of the system startup scripts. Accounting data is kept in **/var/adm** in a file called **acct** or, on later systems, **pacct**. The name was changed to suggest process accounting, not to break all your accounting programs, although that may be a secondary effect. You can theoretically specify a different filename as an argument to **accton**, but if you don't use the default filename, commands such as **lastcomm** may break. **lastcomm** can be used to verify that process accounting is working; it shows every command that has been executed, organized by user or by terminal port.

Process accounting data grows by megabytes per day on a busy system. Each process has a record that includes its UID, elapsed CPU time, average memory use, I/O summary, and several other details. Accounting records are written as each process completes; a program that never terminates does not produce an accounting record.

The **sa** command summarizes CPU data, either by user (typically to the file **/var/adm/usracct**) or by command (to **/var/adm/savacct**). It has a zillion options, most of which have to do with the sort order of the final output and the uninteresting processes to ignore.

Two useful options are **-m** to summarize by user, and **-s** to summarize by command and reinitialize the raw data file. **sa -s** should be run at least daily to control the size of **/var/adm/pacct**.

The user-centric output looks like this:[3]

```
# sa -m
user   #commands    CPU-min  io-operations    memory used
------------------------------------------------------------
root       61038  676.14cpu    41532937tio  77683440k*sec
daemon      8365   27.00cpu     3619385tio    291491k*sec
sys          224    1.26cpu      813849tio     11414k*sec
yiyan        226    2.82cpu      239285tio    242423k*sec
giffard       46    1.41cpu      227312tio    247336k*sec
nikki        187    0.53cpu       35722tio      9419k*sec
...
```

Column one is the login name of the user. Column two is the total number of commands that the user executed. Column three is the CPU time (user plus system) consumed, in minutes. Column four is the total number of I/O operations performed, and column five is the average memory used, measured in kilobyte-seconds.

sa -s summarizes CPU use by command, producing the following output. It also truncates **/var/adm/pacct** to zero length.

3. Accounting commands tend to produce very wide reports. We have in some cases made slight adjustments to fit as much information as possible into the examples.

```
# sa -s
#commands       CPU time              I/O  memory command
-------------------------------------------------------------
198939  992142.01re 4594.91cp     945avio   648k  TOTALS
   163    6818.52re 2915.13cp     960avio    29k  perl
  1808  186951.93re  166.22cp    2871avio  1357k  in.rlogi
    19    1671.62re  163.05cp 2578356avio  4509k  xlock
 19253    2000.08re  124.47cp     332avio   956k  sendmai*
   167    7600.94re   97.71cp   21363avio  3612k  emacs
   401  130627.88re   74.80cp    5101avio  2493k  xterm
  3685  242328.77re   63.87cp     641avio   853k  csh
 28673 -157505.92re   32.77cp      16avio    53k  sh
    43     598.05re   24.74cp   39600avio  5225k  ***other
  1097     166.93re   24.43cp    6928avio  1910k  sendmail
   ...
```

Column one is the number of times the command has been executed. Columns two and three are the real time (wall-clock time) and CPU time (user and system), both in minutes. Column four is the average number of I/O operations per command execution. Column five is the memory usage in kilobytes averaged over the CPU time execution period. Column six is the command name.

The first line contains the totals for all commands. (The word TOTALS isn't really part of **sa**'s output; we added it for clarity.) The ***other entry represents all commands with unprintable characters in their names and all commands that were executed only once.

Command names are truncated to eight characters; an asterisk after a command name means the process was run by root. There are often two entries for common rootly commands: one for regular users executing the command and one for root executing it.

Values are kept in a special floating point format that can wrap around; note the elapsed time entry for **sh**. This example is on a SunOS system where the virtual memory accounting is suspect. 29K average memory usage for **perl** certainly looks low!

The system automatically suspends accounting if the filesystem where accounting data is written becomes too full. When space becomes available on the filesystem again, accounting is restarted.

If the machine crashes or is rebooted, processes that were running are not recorded in the CPU accounting file. A sneaky user could avoid CPU accounting by having a program sleep indefinitely upon completion so that it is always still running when the machine is rebooted.

You might ask, "Who cares about the CPU cycles, memory usage, and so on? Hardware is cheap, and it costs the same whether it's running or

sitting idle." Well, here is an example of using process accounting to tackle a serious breach of security:

When the Internet Worm hit in the Fall of 1988, it created several mischievous processes called **-sh**. A **ps** of the system looked normal, but the load average was going through the roof. Very short-lived processes are hard to spot and understand using **ps**; we ran **sa -s** to summarize the existing accounting data and truncate the raw data file. After five minutes we ran **sa** again and noticed that the **-sh** processes were getting all the CPU time. We then knew what processes to kill to attempt to get things under control again.

Two factors were crucial in our analysis and control of the situation:

- We immediately tried to get (and eventually got) a high-priority root shell using the **nice** command, so our commands did not take minutes to complete when the load average pushed 100.
- Accounting was running so that we could look at the process accounting data. If you choose not to run accounting, it should still be compiled into the kernel so you can turn it on if need be.

Connect-Time Accounting

Connect-time accounting is turned on by the existence of the data file **/var/adm/wtmp**. Login, port, and login/logout times are recorded.

With the advent of window systems and pseudo-terminals, connect-time accounting has become a poor indicator of use. A user logged in to a remote host from several windows is counted multiple times. But a user that accesses the host via **rsh** and runs **xterm** is not counted at all. If you have a workstation in your office and you leave yourself logged in for weeks at a time from several windows, you can record over 100 hours a day of connect time.[4]

The **ac** command summarizes the data in **wtmp** by person (**-p**) or by day (**-d**), and should be run monthly. A quick glance at the output may alert you to unexpected login activity; for example, unexplained use by an employee on vacation or by an inactive account. The **wtmp** file should be rotated or truncated when **ac** is run. (Note that while **sa** truncates the **pacct** file unless told not to, **ac** never truncates **wtmp**.) The per-person summary looks like this:

```
# ac -p
login  connect-hrs
------------------
neves       19.46
weaver       0.13
```

4. Does your site pay overtime?

```
dwight       2.67
jacques      3.51
...
total      759.14
```

A list of login names can also be given. For example:

```
# ac -p evi trent root
login  connect-hrs
------------------
evi         21.03
trent       24.70
root         0.02
total       45.75
```

The output of **ac -d**, which summarizes by date, illustrates that the weekends are periods of lighter usage.

```
# ac -d
date         connect-hrs
------------------------
Nov  1  total    126.70
Nov  2  total    170.98
Nov  3  total    198.19
Nov  4  total    140.47
Nov  5  total    122.83
Nov  6  total     58.92
Nov  7  total     89.93
Nov  8  total    131.55
```

These options can be combined. For example:

```
# ac -dp lane
date         connect-hrs
------------------------
Aug  1  total      8.87
Aug  2  total      7.71
Aug  5  total      4.45
```

The **last** command can be used to determine the actual times that a user logged in and out. Its output can be keyed to a particular user or to a port, which is useful for tracking intruders, finding overworked or broken terminals, and monitoring dial-up modem usage. For example, the command **last lane** shows details on user lane's usage:

```
# last lane
lane  ttype  hartree.cs.colo Thu Aug 5 15:02-19:28 (04:26)
lane  ttyp4  lair.cs.colorad Mon Aug 2 18:44-20:38 (01:54)
lane  ttyp3  lair.cs.colorad Mon Aug 2 18:44-20:38 (01:54)
lane  ttyq8  hartree.cs.colo Mon Aug 2 14:16-16:15 (01:59)
lane  ttyp4  lair.cs.colorad Sun Aug 1 19:17-22:14 (02:57)
lane  ttyp1  lair.cs.colorad Sun Aug 1 19:16-22:13 (02:57)
```

The double entries come from a window system configured to start a remote login session in two windows at once.

```
# last ttyd0
Ufossa     ttyd0            Fri Aug 6 13:44-13:53 (00:09)
Uicarus    ttyd0            Fri Aug 6 13:36-13:36 (00:00)
Ueddie     ttyd0            Fri Aug 6 13:16-13:19 (00:02)
Upathome   ttyd0            Fri Aug 6 12:47-12:48 (00:00)
Upathome   ttyd0            Fri Aug 6 11:45-11:46 (00:00)
```

These are all UUCP hosts; five years ago when home users typically had only ASCII terminals (and when dial-up ports were on workstations rather than concentrated at terminal servers as they are today), this list would have included more real users. For security reasons, it is useful to run **last** on the dial-up ports every so often to look for unusual events (unexpected users or users at unexpected times).

Printer Usage

See Chapter 25 for information about ***/etc/printcap****.*

The line printer daemon **lpd** records printer usage if a printer's entry in **/etc/printcap** has the `af` variable defined and the designated file exists. By convention, printer accounting data files are usually called **/var/adm/***printer***-acct**. They list the number of pages printed for each job, the hostnames where the jobs originated, and the user names of the jobs' owners. The information is summarized by the **pac** command; use it at least monthly. For example:

```
# /etc/pac -Pgutenberg
Login               pages/feet  runs    price
alpo:ross                29.00    2   $  0.58
anchor:zweifel          114.00    2   $  2.28
axon:paul                 7.00    1   $  0.14
blue:brookmak             6.00    6   $  0.12
blue:sanders              1.00    1   $  0.02
columbine:carolyn        44.00   40   $  0.88
columbine:harriet        42.00    7   $  0.84
...
total                   731.00  161   $ 14.62
```

The notation `alpo:ross` refers to user ross on host alpo. The **pac** command takes a flag **-pprice** which sets the price per page in dollars. The default is two cents per page. **pac** does not charge per run, so header pages are not included in the `price` column. There are several other options to **pac**; as always, consult your manual.

It is the responsibility of the printer's input filter to generate accounting records. On PostScript printers, unless the filter actually queries the printer for its page count before and after the job, the page counts are extremely suspect.

Dial-Out Usage

The **tip** and UUCP family of commands (BSD version) record the login name, date, time, phone number, and status of all calls made through a dial-out modem. The information is in **/var/adm/aculog**, which is a text file. Sample log entries are shown below (with lines wrapped to fit).

```
uucp:daemon (Sun Oct 31 18:31:03 1993) <interlink,
  5551234, telebit> call completed
uucp:daemon (Mon Nov  1 08:58:35 1993) <uswestpaging,
  5555678, telebit> call completed
uucp:staff (Mon Nov  1 09:37:07 1993) <t38, , /dev/cua0>
  call completed
uucp:staff (Mon Nov  1 09:38:56 1993) <t38, , /dev/cua0>
  call terminated
```

`interlink` and `uswestpaging` are entries in **/etc/remote** that describe modem characteristics, ports, phone numbers, and so on.

If you allow users to talk directly to a modem via a `dialer` entry in the **/etc/remote** file, then **tip dialer** will circumvent accounting, since **tip** does not directly cause the phone number to be dialed and therefore cannot write an appropriate log entry.

There are no standard tools that summarize the **aculog** file, but **grep** can be used to identify long distance calls. The phone company usually provides itemized bills for long distance calls that can be matched against your **tip** log files.

Unfortunately, most modems can be put into command mode via a special sequence of characters and pauses. A thief can circumvent accounting by calling a local number, sending the modem's break sequence, and then instructing the modem to hang up and redial unbeknownst to **tip**. The break sequence is usually configurable, so you might want to make it something nonstandard if you allow general access to modems.

Summaries

Table 28.1 details the files that store accounting data on BSD-ish systems. The owner, group, and mode (shown here in octal) of the accounting data files are important; any program used to reinitialize these files should be sure to **chown**, **chgrp**, and **chmod** appropriately.

Table 28.2 summarizes the BSD accounting commands.

28.6 ACCOUNTING UNDER ATT

Most versions of ATT UNIX contain a fairly complete accounting system implemented with C programs and **sh** scripts. These scripts are typically run daily and monthly by **cron**. All accounting is done under the

Table 28.1 Summary of BSD accounting files

Data	Filename	Type	Owner	Group	Mode
CPU, Memory	**acct** or **pacct**	Binary	root	system	644
Connect time	**wtmp**	Binary	root	system	644
Printer usage	*lp*-**acct**	Text	daemon	daemon	644
Dial-out usage	**aculog**	Text	uucp	daemon	660

Table 28.2 Summary of BSD accounting commands

Data	Command	Frequency to run
CPU, Memory	**accton**	During boot
	sa	At least daily
	lastcomm	As needed
Connect time	**ac**	Monthly
	last	As needed
Printer usage	**pac**	Monthly for each printer

login adm with home directory **/var/adm**; accounting information is kept in **/var/adm/acct**. The accounting programs and scripts are usually in **/usr/lib/acct**.

Setting Up Accounting

On most systems, accounting does not run by default, so you must set it up. The accounting programs may even be unbundled (either sold separately or included with standard distribution but separately installed).

To see if you have the accounting system on-line, look in the directory **/usr/lib/acct**. If it's empty, go back to the distribution CD-ROM. Otherwise, make sure that the adm login exists. If it does not, you can create it by adding the following line to **/etc/passwd**:

```
adm:*:4:4:Administrative Login:/var/adm:/bin/sh
```

adm traditionally has UID four, but that is arbitrary. It's best to look on the distribution and observe the UID of all the accounting programs and configuration files. After creating the adm login, make sure that the directory **/var/adm/acct** exists and is owned by adm. **/var/adm** should include a **.profile** file with the following contents:

```
PATH=/usr/lib/acct:/bin:/usr/bin
```

You should also create the subdirectories **night**, **sum**, and **fiscal** in **/var/adm/acct**.

Accounting information is not gathered until the **startup** command is executed. To start accounting automatically on a system using BSD-style

/etc/rc files, such as HP-UX, add the following line to the appropriate section of the **rc** file:

```
/bin/su - adm -c /usr/lib/acct/startup
```

The **/usr/lib/acct/shutacct** command turns off accounting; some systems need to have it done specifically during the shutdown process, and some do it automatically.

For systems using ATT-style **init** levels to specify the actions to take when going multi-user (for example, Solaris), the following shell script added to **/etc/init.d** and hard-linked to the **rc2.d** directory would turn accounting on and off as the machine came in and out of multi-user mode.

See Chapter 2, *Booting and Shutting Down*, for more information about starting services at boot time.

```
#!/bin/sh
# rc script to start and stop accounting

case "$1" in
'start')
    if [ -x /usr/lib/acct/startup ] ; then
        echo "accounting started."
        /usr/lib/acct/startup
    fi
    ;;

'stop')
    if [ -x /usr/lib/acct/shutacct ] ; then
        /usr/lib/acct/shutacct
    fi
    ;;

*)
    echo "Usage: /etc/init.d/acct { start | stop }"
    ;;
esac

exit 0
```

A copy of this script is included on the CD-ROM. IRIX 4.0 includes a similar script, but IRIX 5.2 and Solaris 2.4 do not. Sometimes the script exists in **/etc/init.d** but the link to **rc2.d** is missing.

In order for the system to charge properly for system usage, you must tell it which hours you consider "prime time" and which days you consider holidays. This is done in the file **/usr/lib/acct/holidays**. A sample **holidays** file follows.

```
* Prime/Nonprime Table for Accounting System
*
* Year   PrimeStart  NonPrimeStart
  1994      0900         1800
*
    1       Jan 1        New Year's Day
   35       Feb 4        AT&T vs. UC Lawsuit Settled
  231       Aug 19       ucbvax Retired
  323       Oct 31       Halloween
  325       Nov 2        Anniversary of the Internet Worm
  359       Dec 25       Christmas Day
```

All lines starting with a star are comments. The first non-comment line specifies the current year (in this case 1994), and the start and end of prime hours. In this example, prime hours are from 9:00 a.m. to 6:00 p.m. Prime time must be a contiguous block of time; all other times are considered non-prime. There is no way to have more than two classes of service (prime and non-prime) or to have more than one block of prime time per day (for example, prime time from 10:00 a.m. to 5:00 p.m. and again from 7:00 p.m. to 10:00 p.m.).

The rest of the lines in the file list the days that you consider holidays, which are treated the same as weekends. The fields are:

```
yearday monthday description
```

Both a C program and a ***perl*** *script to compute Julian dates are included on the CD-ROM.*

Of these, only *yearday* (the Julian date) is actually used by accounting programs. Obviously, the **holidays** file must be updated each year. If accounting is run with an out-of-date **holidays** file, a mail message will be sent to users adm and root, and log entries will be written. If you don't use accounting to charge people for computing services, don't worry too much about the contents of the holidays file. It only needs to contain the line specifying the current year and prime time hours in order to keep the log files from filling with error messages.

See Chapter 10 for more information about ***cron****.*

Running **startup** tells the system to start archiving accounting data, but it does not cause the data to be processed. The following **cron** entries, usually placed in adm's crontab with **crontab -e adm**, support both daily and monthly processing of accounting. They run accounting in the wee hours of the night, when users won't be disturbed.

```
# daily and weekly accounting chores
#
0 4 * * * /usr/lib/acct/runacct 2>/var/adm/acct/nite/d2log
0 1 * * 4 /usr/lib/acct/dodisk
0 * * * * /usr/lib/acct/ckpacct
#
# monthly accounting
#
0 2 1 * * /usr/lib/acct/monacct
```

What Accounting Does

The **runacct** program generates several files that contain daily accounting information and stores them in **/var/adm/acct/sum**. The only files that are of any real interest are the report files.

There are report files for each day since **monacct** was last run called **rprt***mmdd* where *mm* is the month and *dd* is the day. These reports can be printed out if you want a record of daily accounting. The reports contain summaries of terminal usage, command usage, disk usage, and time of last login. The format of the reports is self-explanatory.

If **runacct** does not run to completion because the system crashes, it must be restarted by hand. The manual page gives a complete description of how to restart **runacct** so it picks up where it left off. **runacct** writes error messages to **/var/adm/acct/nite/fd2log** (or whatever file you specify in the **cron** entry that starts **runacct**). For example:

```
acctcms: Hash table overflow. Increase CSIZE
***UPDATE /etc/holidays WITH NEW HOLIDAYS***
acctprc2: INCREASE A_USIZE
```

Unfortunately, only the **holidays** file can be updated by mere mortals. CSIZE is the size of a hash table defined in **acctcms.c**; it cannot be changed unless you have source code, and its value is usually 1,000. A_USIZE is the maximum number of logins and is set in **acctdef.h**, usually to 500; it also requires access to the source code to change. Solaris has boosted these constants to more reasonable values.

Error messages are also written to **/var/adm/acct/nite/log***mmdd*, where *mm* is the current month and *dd* the current day. For example, the file **log0601** contains

```
***UPDATE /etc/holidays WITH NEW HOLIDAYS***
acctcon1: RECOMPILE WITH LARGER A_TSIZE
```

Again, you must have source code to fix most errors.

The **dodisk** program collects disk usage information. In the example above, **dodisk** is only run once a week on Thursdays; it uses a bit of CPU and I/O bandwidth, and for many users the results are not very volatile. The data is stored in **/var/adm/acct/nite/disktacct** and is merged into the file **daytacct** by the **runacct** script.

*See page 620 for more information about **du**.*

The **ckpacct** command monitors the process accounting data file **pacct** (in either **/usr/adm** or **/var/adm**) and splits it up when it gets larger than a certain size, usually 1,000 **du**-sized blocks. **ckpacct** also monitors free space on **/usr** or **/var** and disables accounting if there are fewer than 500 free blocks.

Such monitoring makes sense if the monitor watches the filesystem on which the **pacct** file lives. Under SunOS, **/usr** is watched, but **pacct** is in **/var/adm**, usually on a separate partition. Fortunately, **ckpacct** is a script and so you can change **/usr** to **/var** to fix this problem.

monacct summarizes the daily reports for the previous month, stores summaries in **/var/adm/acct/fiscal/fiscrpt***mm* (where *mm* is again the month), and restarts the summary files in the **sum** directory. If you want to charge users for their usage, you can write a simple **perl** or shell script that examines the monthly summaries and generates invoices for each user.

In addition to the information that is collected automatically when accounting is turned on, the **chargefee** program allows you to assess additional fees on specific users. This is useful if you want to charge for something that you did for them, such as loading a tape. You could use the command

```
/usr/lib/acct/chargefee joe 10
```

to charge the user joe for ten accounting units (the meaning of units is completely arbitrary). All fees that are charged with **chargefee** will appear in the report files, so you can include them in invoices.

Ideally, you should not have to worry about the nitty-gritty of how data files are processed. However, if you want to modify the way that accounting works, or if something is not working correctly, you may have to investigate its inner workings. The accounting system keeps a lot of internal files in **/var/adm/acct/nite** and **/var/adm/acct/sum**. Descriptions of the various commands and data files can be found in the documentation. Table 28.3 (next page) describes the files usually found in the directory **/usr/lib/acct**. Entries with a program in parentheses at the end of the description field are not run directly, but rather by the designated program.

This is an impressive array of accounting artillery. However, very few of the commands are used directly by a system administrator. Daily and monthly summaries are normally handled out of **cron**. Occasionally, accounting will fail to run to completion, but the ATT accounting system does a good job of letting you restart it.

runacct produces the daily summaries. It is usually invoked so that its error output is sent to **/var/adm/acct/nite/fd2log** (file descriptor two is the standard error channel). If there is something wrong with accounting, diagnostic messages will show up in this file. If you ignore accounting and have something misconfigured, the file can grow quite quickly. Keep an eye on it. Also, periodically inspect the other output files beneath **/var/adm/acct**.

Table 28.3 Summary of ATT accounting files in /usr/lib/acct

	Command	Type	Description
Administration	**startup**	**sh** script	Runs at boot time to enable accounting
	acctон	Program	Turns on process accounting
	turnacct	**sh** script	Turns on accounting to **/usr/adm/pacct**
	shutacct	Program	Turns off accounting and logs to **wtmp**
	chargefee	**sh** script	Charges specific users
	holidays	Text file	List of holidays
	nulladm	**sh** script	Reinitializes files and checks ownerships
	remove	**sh** script	Cleans up **/usr/adm/acct/sum**
Time	**acctcon1**	Program	ASCIIfies connect-time records (**runacct**)
	acctcon2	Program	Converts to **tacct** format (**runacct**)
	lastlogin	**sh** script	Updates last login record in **sum/loginlog**
	acctwtmp	Program	Adds boot record to **wtmp** file
	fwtmp	Program	Fixes dates in **wtmp** when changed by **date**
	wtmpfix	Program	Recognizes/repairs a bad **wtmp** file
CPU	**ckpacct**	**sh** script	Restarts **pacct** file if it is too big
	acctcms	Program	Makes command usage records (**runacct**)
	acctprc1	Program	Makes process records (**runacct**)
	acctprc2	Program	Makes process records (**runacct**)
Disk use	**acctdisk**	Program	Makes disk usage records (**dodisk**)
	acctdusg	Program	Makes disk usage records (**dodisk**)
	diskusg	Program	Generates disk accounting data by user
	dodisk	**sh** script	Takes a snapshot of disk usage
Reports	**runacct**	**sh** script	Summarizes daily data
	acctmerg	Program	Merges processed records (**runacct**)
	prctmp	**sh** script	Prints session record file
	prdaily	**sh** script	Prints previous day's accounting summaries
	prtacct	**sh** script	Prints accounting records from **tacct** files
	monacct	**sh** script	Produces monthly reports

Printer Accounting

See Chapter 25 for more information about printing.

ATT systems do not have printer accounting, although the spooler records print jobs in the file **/var/spool/lp/logs/requests**. The number of bytes printed is recorded, but not the number of pages, so the information is essentially useless for accounting purposes. If you want true printer accounting like that provided by the BSD **pac** command, it must be built into the printer interface scripts used in the **lp** system.

28.7 Specifics for Various Operating Systems

Most vendors use BSD- or ATT-style accounting without much modification. Tables 28.4 and 28.6 detail the locations of accounting data files and the commands supplied by vendors to administer them.

Table 28.4 File and command locations by vendor (ATT)

File	Solaris	HP-UX	IRIX
ATT commands	**/usr/lib/acct**	**/usr/lib/acct**	**/usr/lib/acct**
ATT reports	**/var/adm/acct**	**/usr/adm/acct**	**/var/adm/acct**
acct or **pacct**	**/var/adm**	**/usr/adm**	**/var/adm**
wtmp	**/var/adm**	**/etc**	**/var/adm**
pac	–	–	**/usr/etc**

Solaris is straight ATT, with no **sa**, **ac**, or **pac** to be found. There is no **/usr/etc** either. The holidays file is in **/etc/acct**, carefully moved to its own directory so that any old accounting scripts expecting it to be in **/usr/lib/acct** will break without warning.

Table 28.5 describes four bonus commands in **/usr/lib/acct** that join those from Table 28.3. The last two of these commands account for users who are logged in while accounting is being run.

Table 28.5 Bonus commands in Solaris

Command	Type	Description
acctcon	Program	Combines **acctcon1** and **acctcon2** programs
acctprc	Program	Combines **acctprc1** and **acctprc2** programs
closewtmp	Program	Fakes entries for users logged on during accounting
utmp2wtmp	Program	Adds **wtmp** records for users currently logged on

HP-UX uses ATT-style accounting and 4.3BSD-style logging via **syslog**. Accounting is a separate subsystem and must be explicitly loaded from the software distribution. Commands are in **/usr/lib/acct**; raw data files are in **/usr/adm** with summaries in **/usr/adm/acct**. Accounting must be started in **/etc/rc** to begin collecting data.

IRIX uses ATT-style accounting with commands in **/usr/lib/acct** and the data summarized in **/var/adm/acct**. It supports both ATT and BSD printing; the **pac** printer accounting program is included.

IRIX distinguishes between an empty data file for a configured service and an unconfigured service. The directory **/etc/config** contains files representing each service; the command **chkconfig** checks to see if the service is configured or not. If **acct** contains the word on and crontab entries are as below, accounting data will be collected and summarized.

```
# runacct processes connect, fee, disk, and process
# accounting files
0  4  *  *  1-6   if /etc/chkconfig acct; then
  /usr/lib/acct/runacct 2> /usr/adm/acct/nite/fd2log; fi
```

```
# ckpacct checks the size of /usr/adm/pacct
5  *  *  *  1-6   if /etc/chkconfig acct; then
  /usr/lib/acct/ckpacct; fi
# monacct creates summary files in /usr/adm/acct/fiscal
0  5  1  *  *   if /etc/chkconfig acct; then
  /usr/lib/acct/monacct; fi
```

Table 28.6 File and command locations by vendor (BSD)

File	SunOS	OSF/1	BSDI
accton	**/usr/lib/acct**	**/usr/sbin/acct**	**/usr/sbin**
sa	**/usr/etc**	**/usr/sbin**	**/usr/sbin**
acct or **pacct**	**/var/adm**	**/var/adm**	**/var/account**
usracct	**/var/adm**	**/var/adm**	**/var/account**
savacct	**/var/adm**	**/var/adm**	**/var/account**
ac	**/usr/etc**	**/usr/sbin**	–
wtmp	**/etc**	**/var/adm**	**/var/log**
pac	**/usr/etc**	**/usr/sbin**	**/usr/sbin**
ATT reports	**/usr/lib/acct**	**/var/adm/acct**	–
ATT commands	**/var/adm/acct**	**/usr/sbin/acct**	–

SunOS uses ATT-style accounting but also includes the BSD commands for login, process, and printer accounting. As shipped, data files are in **/var/adm/acct** and programs and scripts are in **/usr/lib/acct**. Accounting must be configured into the kernel in order for process accounting data to be kept. The variable SYSACCT must be defined; see Chapter 13, *Configuring the Kernel*.

The accounting data files were traditionally beneath **/usr/adm**, which Sun has linked to **/var/adm**. SunOS has an additional accounting command not in the general ATT list from the previous section: **getname**. **getname** is an undocumented command that looks up a user in **/etc/passwd**. It is used by several scripts in **/usr/lib/acct**.

OSF/1 uses ATT-style accounting, but has moved the commands from **/usr/lib/acct** to **/usr/sbin/acct**. DEC's OSF/1 2.0 includes the BSD accounting commands in **/usr/sbin**. Older versions had the man pages but not the commands (except for **pac**).

Accounting on BSDI is BSD-style with commands in **/usr/sbin**; the **ac** command is missing. Process accounting information is archived in **/var/account/acct**. Login information is in **/var/log/wtmp**.